Early Consensus in Message-passing Systems Enriched with a Perfect Failure Detector and its Application in the Theta Model

François Bonnet, Michel Raynal

Publications Internes de l'IRISA
ISSN : 2102-6327
PI 1937 – octobre 2009

Early Consensus in Message-passing Systems Enriched with a Perfect Failure Detector and its Application in the Theta Model

François Bonnet[*] , Michel Raynal[**]
francois.bonnet@irisa.fr, raynal@irisa.fr

Abstract: While lots of consensus algorithms have been proposed for crash-prone asynchronous message-passing systems enriched with a failure detector of the class Ω (the class of *eventual leader* failure detectors), very few algorithms have been proposed for systems enriched with a failure detector of the class P (the class of *perfect* failure detectors). Moreover, (to the best of our knowledge) the *early decision and stopping* notion has not been investigated in such systems.

This paper presents an early-deciding/stopping P-based algorithm. A process that does not crash decides (and stops) in at most $\min(f+2, t+1)$ rounds, where t is the maximum number of processes that may crash, and f the actual number of crashes ($0 \leq f \leq t$). Differently from what occurs in a synchronous system, a perfect failure detector notifies failures asynchronously. This makes the design of an early deciding (and stopping) algorithm not trivial. Interestingly enough, the proposed algorithm meets the lower bound on the number of rounds for early decision in synchronous systems. In that sense, it is optimal.

The paper then presents an original algorithm that implements a perfect failure detector in the Theta model, an interesting model that achieves some form of synchrony without relying on physical clocks. Hence, the stacking of these algorithms provides an algorithm that solves consensus in the Theta model in $\min(f+2, t+1)$ communication rounds, i.e., in two rounds when there are no failures, which is clearly optimal.

Key-words: Asynchronous message-passing system, Consensus problem, Early decision, Perfect failure detector, Process crash, Theta-model

---

Consensus avec Décision au plus Tôt dans les Systèmes à Passage de Messages enrichis par un Détecteur de Fautes Parfait et ses Applications dans le Modèle Theta

Résumé : *Cet article présente un algorithme pour le problème du consensus avec la propriété de décision au plus tôt. Il propose ensuite un algorithme implémentant un détecteur de fautes parfait dans le modèle Theta.*

Mots clés : *Système asynchrone à passage de messages, Problème du consensus, Décision au plus tôt, Détecteur de fautes parfait, Panne de processus, Modèle Theta*

---

[*] Projet ASAP : équipe commune avec l'INRIA, l'INSA et université de Rennes 1
[**] Projet ASAP : équipe commune avec l'INRIA, l'INSA et université de Rennes 1

1 Introduction

The consensus problem The consensus problem has a very simple formulation. Each process is assumed to propose a value, and each non-faulty process has to decide a value (termination), such that a decided value is a proposed value (validity), and no two distinct processes decide different values (agreement).

Consensus is central in the design of distributed software as, in any distributed application, in one way or another, the processes have to coordinate and agree on some common values. The most well-known example of its use is the *total order broadcast* problem (also named atomic broadcast) in which the processes have to deliver in the same order the messages they broadcast. This is typically, a communication problem plus an agreement problem (the processes have to agree on the same delivery order) that can be solved as soon as one can solve consensus.

Consensus in synchronous systems Consensus can easily be solved in synchronous message-passing systems despite the crash of any number of processes. In such a system, the processes proceed by consecutive asynchronous rounds that they execute simultaneously. During a round, a process broadcasts a message, receives messages -sent at the very same round- and executes local computation.

Let t be the maximum number of processes that may crash, $0 < t < n$, where n is the number of processes. The algorithms that solve consensus in synchronous systems require at least $t+1$ rounds in the worst case (i.e., when t processes crash) [8]. Failures do occur, but are rare. Hence, the notion of *early decision* has been developed [6]. In a run, the number of rounds needed for agreement has to be related to the actual number of crashes (denoted f, $0 \leq f \leq t$), and not to the maximal number of processes that may crash. Synchronous consensus algorithms have been designed where the processes decides in at most $\min(f+2, t+1)$ rounds, which has been shown to be optimal [6].

Consensus in asynchronous systems Differently from synchronous systems, there is no bound on processing time and message transfer delays in an asynchronous system. Hence, the round notion is not given for free. The processes can construct it, but there is no guarantee that they are executing simultaneously the same round.

When we consider the consensus problem, the most important result is its impossibility to be solved in an asynchronous system despite even only one process crash. This is the celebrated FLP result [7].

Among the approaches that have been proposed to circumvent this impossibility, the *failure detector* approach consists in enriching the asynchronous system with a module per process, that gives it hints on which processes have failed [4]. According to the quality of these hints, several failure detector classes can be defined.

It has been shown that the class denoted Ω is the weakest class that allows solving consensus in asynchronous message-passing systems where $t < n/2$. This class is the class of *eventual leader* failure detector. It provides each process p_i with a read-only local variable $leader_i$ that always contains a process identity. These variables are such that there is a finite (but unknown) time after which they all contain forever the same process identity, and this identity refers to a non-faulty process.

Ω-based consensus algorithms, for asynchronous systems where $t < n/2$, have designed (e.g., [9, 10, 12, 13, 15]). The number of rounds generated by these algorithms depends naturally on the behavior of Ω, namely, the time instant after which the modules output the same identity of a non-faulty process. If the non-faulty leader is elected from the very beginning, two rounds are necessary and sufficient [11].

The class P of perfect failure detectors A failure detector of the class P provides each process p_i with a set denoted $suspected_i$ that (a) eventually contains the identities of all the processes that crash, and (b) never contains the identity of a process before it crashes.

The respective power of synchronous systems and asynchronous systems enriched with P has been investigated in [5], where is presented an asynchronous P-based algorithm where the processes decide in $t+1$ rounds.

Content of the paper This paper is centered on the class P of failure detectors, in the context of message-passing systems. It presents and proves correct two new algorithms.

- The first is a P-based early-deciding/stopping consensus algorithm. More precisely, a process decides and halts in at most $\min(f+2, t+1)$ rounds. Hence, this algorithm attains the synchronous system lower bound. As a synchronous system is stronger than an asynchronous system enriched with P, the proposed algorithm is consequently optimal with respect to the number of rounds.

  To our knowledge, this is the first P-based early-deciding consensus algorithm. It is important to notice that the design of such an algorithm is not trivial. This is because two processes can be at distinct rounds at the same time (which cannot occur in a synchronous system). Consequently a process executing round r can be notified during that round of the crash of another process that is executing round $r+1$.

- The second algorithm that is presented implements a failure detector of the class P in an instance of the Theta model [2, 14, 18]. A very attractive feature of this model lies in the fact that it captures synchrony properties without using physical clocks (and associated timers). Basically, it assumes that there is a bound θ on the ratio of the upper bound and the lower bound on message transit delays. Those have not to be known, and can even vary with time.

  The proposed θ-based perfect failure detector algorithm is based on the notion of *fair communication*, a notion that is on algorithms and not on the underlying system [1].

Interestingly, the stacking of these two algorithms provides us with a consensus algorithm for the asynchronous message-passing distributed computing model in which a process decides and stops in at most $\min(f+2, t+1)$ rounds, which means in a bounded number of rounds. This is a very attractive feature, as it provides the application processes with a known bound. (Let us remember that such a bound cannot be guaranteed when using an eventual leader failure detector.)

Roadmap The paper is made up of 5 sections. Section 2 presents the base computation asynchronous model and a formal definition of the class P. Then, Section 3 presents and proves correct the early-deciding P-based consensus algorithm. Section 4 presents and proves the θ-based algorithm that builds a failure detector of the class P. Finally, Section 5 concludes the paper.

2 Distributed system model

2.1 Crash-prone asynchronous message-passing system

Processes The system is made up of n processes denoted $p_1, \ldots, p_n$. Each process is asynchronous, which means that any (finite) time duration can elapse between any two consecutive steps of a process.

Communication The processes communicate by exchanging messages through channels. Each pair of processes is connected by a bi-directional channel. Channels are asynchronous, in the sense that message transit times are finite but not bounded. Moreover channels are reliable: they do not create, corrupt or lose messages. They are not required to ensure the "first in first out" property.

Failure model Up to t processes may crash, $0 < t < n$. A process executes correctly its algorithm until it crashes (if it ever crashes). After it has crashed, a process does nothing (it is "dead"). Given a run, f denotes the number of processes that crash in that run, hence $0 \leq f \leq t$.

Notation The previous type of systems is denoted $\mathcal{AS}_{n,t}[\emptyset]$ ($\emptyset$ means that the asynchronous system is "pure", i.e., not enriched with additional assumptions).

2.2 The class P of perfect failure detectors

Definitions The following definitions are from [3, 4]. A failure pattern is a function $F()$ such that $F(\tau)$ is the set of processes that have crashed up to time τ (included). Given a failure pattern F, $Faulty(F) = \lim_{\tau \to +\infty} F(\tau)$ is the set of processes that are faulty in that failure pattern. $Correct(F) = \{1, \cdots, n\} \setminus Faulty(F)$ denotes the set of processes that do not crash in that failure pattern. Finally, let $Alive(F, \tau)$ the set of processes that are not crashed at time τ in the failure pattern $F()$, i.e., the set $\{1, \cdots, n\} \setminus F(\tau)$.

Informally, a failure detector is a device that provides each process p_i with a local variable that p_i can only read and that gives it hints on failures. Formally, a failure detector is defined from a failure detector history function $H()$. $H(i, \tau)$ is the value output at p_i by the failure detector at time τ. When it reads $H(i, -)$, p_i obtains its current content.

According to the "quality" of the hints obtained from $H(i, -)$, several classes of failure detectors can be defined.

The class of perfect failure detectors This class, denoted P, contains the failure detectors that provide each process p_i with a set $suspected_i$ that satisfies the following properties. The value of $suspected_i$ at time τ (namely, $H(i, \tau)$) is denoted $suspected_i^\tau$.

- Completeness. $\exists \tau : \forall \tau' \geq \tau,\ \forall i \in Correct(F), \forall j \in Faulty(F) : j \in suspected_i^{\tau'}$.
- Strong accuracy. $\forall\ \tau :\ \forall\ i, j \in Alive(F, \tau) :\ j \notin suspected_i^\tau$.

The completeness property is an *eventual* property that states that there is a finite but unknown time after which any faulty process is suspected by any non-faulty process. The strong accuracy property is a *perpetual* property that states that no process is suspected before it crashes.

Notation The system model $\mathcal{AS}_{n,t}[\emptyset]$ enriched with a failure detector of the class P is denoted $\mathcal{AS}_{n,t}[P]$.

2.3 Synchronous system vs asynchronous system enriched with P

Synchronous systems In a round-based synchronous system the round notion is given for free. The processes progress in a lock-step manner: they all execute the same round at the same time. During a round r a process executes the following steps.

- First p_i sends a round r message to the other processes. If it crashes while executing this step, an arbitrary subset of processes receive its round r message.
- Then, p_i waits for the round r messages from the other processes.
- And finally p_i executes local computation.

The progress from a round r to round $r+1$ is not managed by each process separately (as done in an asynchronous system), but governed by the system that informs the processes of the passage from r to $r+1$. The most important feature of a synchronous system is that a message sent in a round is received in the very same round. If follows that if a process p_i does not receive a round r message from a process p_j, it can safely conclude that p_j has crashed: the absence of message indicates a process crash.

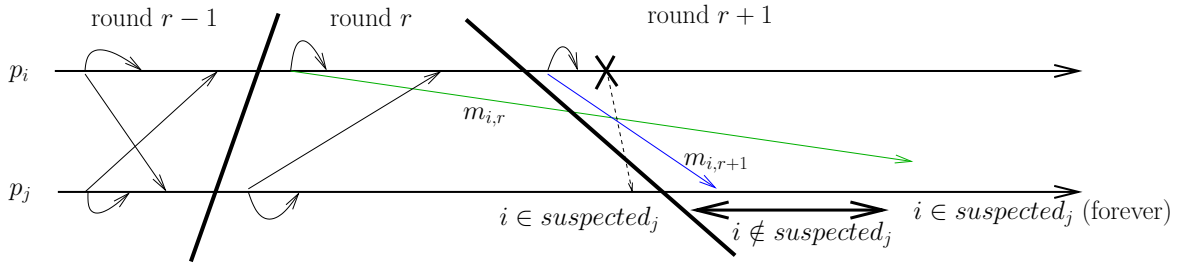

Figure 1: Rounds, asynchrony and perfect failure detector

A bad phenomenon due to asynchrony In order to point out differences between a synchronous system and an asynchronous system enriched with P, let us consider Figure 1 where the round boundaries are indicated by bold lines. There are two processes p_i and p_j. During round $(r-1)$, each receives two messages, and proceeds to round r. During round r, p_i sends the message $m_{i,r}$, receives two messages, and proceeds to round $(r+1)$. Moreover, in round $(r+1)$, p_i sends the message $m_{i,r+1}$ and crashes. On its side, during round r, after having sent a message to p_i, the process p_j waits for a message from p_i. But, in the scenario described on the figure, the corresponding message $m_{i,r}$ is very slow (asynchrony), and p_j is informed of the crash of p_i before receiving $m_{i,r}$. As P provides it with safe suspicions, p_j stops waiting and proceeds to round $(r+1)$.

Then during round $(r+1)$ the following can happen. Let us remember that any failure detector of the class P ensures that no process is suspected before it crashes (accuracy property), and if a process p_i crashes, there is a time after which i remains forever in $suspected_j$ (completeness property). This does not prevent the scenario described in the figure that depicts a finite period during which i does not belong to $suspected_j$. This period starts when i is added to $suspected_j$ for the first time, and ends when it is added to $suspected_j$ for the last time. If this occurs, p_j does not suspect p_i when it receives the message $m_{i,r+1}$ (and consequently considers that message): p_j has then received from p_i a round $(r+1)$ message and no round r message.

This bad scenario can never occur in a synchronous system. (In a synchronous system the round boundary lines are "orthogonal" with respect to the process axes.) The message $m_{i,r}$ would be received by p_j during round r, and the message $m_{i,r+1}$ during round $(r+1)$. In the crash pattern presented in the Figure, p_j would discover the crash of p_i during round $(r+2)$: as it will not receive a round $(r+2)$ message from p_i, pj will conclude that p_i has crashed at round $(r+1)$ or $(r+2)$.

Hiding the bad phenomenon In order to prevent this bad scenario from occurring, each process p_i can manage a set $crashed_i$ (that is initially empty). Differently from the set $suspected_i$ it is provided with by the failure detector, $crashed_i$ is monotonous, i.e., $\forall \tau, \tau' : (\tau \leq \tau') \Rightarrow (crashed_i^{\tau} \subseteq crashed_i^{\tau'})$ (where $crashed_i^{\alpha}$ is the value of $crashed_i$ at time α).

To implement $crashed_i$, p_i can use a thread that forever executes $crashed_i \leftarrow crashed_i \cup suspected_i$. It is easy to see, that the sets $\{crashed_i\}_{1 \leq i \leq n}$ of the processes satisfy the completeness and accuracy properties of P plus the monotonicity

property that rules out the bad phenomenon described in Figure 1. In the following, we consider the sets $crashed_i$ instead of the outputs $suspected_i$ provided by a perfect failure detector.

3 Two P-based consensus algorithms

3.1 An algorithm without early decision

Underlying principle The proposed consensus algorithm (described in Figure 2) is based on the following principle that consists in directing each process to broadcast its current estimate during each round. Moreover, a deterministic function is used to select the decided value (e.g., a process decides the smallest value it has ever seen).

Local variables

- est_i: p_i's current estimate of the decision value. Its initial value is v_i (value proposed by p_i).
- r_i: p_i's current round number. Its initial value is 1.
- Let us remember that p_i can read $crashed_i$ that is a monotonous version of $suspected_i$.

The scope of the other auxiliary local variables is one round. Their meaning follows clearly from their identifier.

```
operation propose (v_i):
(1)  est_i ← v_i; r_i ← 1;
(2)  while r_i ≤ t + 1 do
(3)      begin asynchronous round
(4)      broadcast EST (r_i, est_i);
(5)      wait until (∀ j ∉ crashed_i: (EST (r_i, −) received from p_j));
(6)      let rec_from_i = {1, ..., n} \ crashed_i;
(7)      let est_rec_i = {est from the processes in rec_from_i};
(8)      est_i ← min(est_rec_i);
(9)      r_i ← r_i + 1
(10)     end asynchronous round
(11) end while;
(12) return (est_i).
```

Figure 2: Consensus algorithm for $\mathcal{AS}_{n,t}[P]$ (code for p_i)

Behavior of a process As in a round-based synchronous system, the behavior of a process p_i during a round r is made up of two phases, a communication phase followed by a local computation phase.

- Communication phase. A process p_i first broadcasts EST (r_i, est_i) to inform the other processes on its current state. Then, it waits until it has received a message EST $(r_i, -)$ from each process p_j but the processes that, to its knowledge, have crashed. Let rec_from_i denote the set of processes from which p_i considers and processes the messages it has received.
- Computation phase. Then p_i computes its new estimate value, that is the smallest value carried by a message it has received from a process in rec_from_i.

3.2 Proof of the algorithm

Notation Let $min(r)$ denote the smallest value of the current estimates of the processes that terminate round r (i.e., after the updates of est_i during round r). Moreover, $min(0)$ denotes the smallest proposed value.

Theorem 1 *The algorithm described in Figure 2 solves consensus in $\mathcal{AS}_{n,t}[P]$. Moreover the decided value is $min(t)$.*

Proof
Proof of the validity property. This property follows trivially from the following observations: (a) a process decides the current value of its estimate, (b) the initial values of the estimates are the proposed values, and (c) when updated, a local

estimate is set to the minimum of a set of current estimates it has received.

Proof of the terminaison property. Due to the completeness property of the failure detector, a process can never be blocked forever in the **wait** statement of line 5. Consequently all correct processes terminate round $t+1$ and decide.

Proof of the agreement property. In order to establish this property, we first prove a claim that captures the synchronization provided by the net effect of the **wait** statement and the use of a class P failure detector.

Claim C1. For any r, $1 \leq r \leq t+1$, when a process p_i terminates round r (this occurs when it sets r_i to $r+1$), there is no alive process in a round $r' < r$. (At any time, any two alive processes are separated by at most one round.)
Proof of the claim C1. The claim follows from the accuracy property of P. If p_j is alive and executing round $r-1$ or a smaller round (or is alive and has not yet started if $r = 0$), we have $j \notin suspected_i$, which implies $j \notin crashed_i$. It then follows from the predicate "$\forall j \notin crashed_i$: (EST $(r_i, -)$ received from p_j" (**wait** statement) that controls the progress of p_i from r to $r+1$, that p_i cannot proceed to $r+1$. End of proof of claim C1.

The proof of the agreement property is by contradiction. Let us suppose that some process decides a value different from $min(t)$. We show that there are then $(t+1)$ crashes, which contradicts the model definition. To that end, let us consider the following definitions where r is any round number such that $1 \leq r \leq t+1$.

- $Q(r) = \{\ p_x \mid \text{such that } est_x \leq min(t) \text{ when } p_x \text{ starts round } r\ \}$.
- $R(r) = Q(r) \setminus (R(0) \cup R(1) \ldots R(r-1))$ with $R(0) = \emptyset$. $R(r)$ is the set of processes p_x that start round r with $est_x \leq min(t)$ but did not start a round $r' < r$ with $est_x \leq min(t)$.

By a slight abuse of language, we say that, when a process decides, it is executing the communication-free round $t+2$. Hence, a process that crashes after having decided crashes during the round $t+2$.

Claim C2. $\forall r:\ 1 \leq r \leq t+1$: (i) $|R(r)| \geq 1$, and (ii) all the processes in $R(r)$ crash while executing round r or $(r+1)$.

It follows from claim C2 that, for each round r, $1 \leq r \leq t+1$), at least one process crashes during r or $r+1$, from which we conclude that $(t+1)$ processes crash. This contradicts the fact that at most t processes may crash and proves consequently the agreement property.

Proof of claim C2. The proof is by induction on r. Let us first consider the base case $r = 1$.

- Proof of (i): $|R(1)| \geq 1$. As $min(t)$ is the smallest estimate value at the end of round t, and the algorithm does not create estimate values, there is at least one process whose proposed value is equal to $min(t)$.
- Proof of (ii): all the processes in $R(1)$ crash while executing round 1 or round 2. Let us assume by contradiction that there is a process $p_i \in R(1)$ that does not crash while executing round 1 or round 2 (hence, p_i terminates round 2). Due to the claim C1, there is no process alive in round 1 when p_i terminates $r = 2$. It follows that all the processes that have terminated the first round have received and processed the message EST $(1, est_i)$, with $est_i \leq min(t)$. Consequently, all the processes p_j that enter the second round have updated their estimate est_j to a value $\leq min(t)$ before entering the second round. Since estimates can only decrease during the execution, this implies that any process p_j that starts the last round has an estimate $est_j \leq min(t)$. By the very definition of $min(t)$ it means that such an estimate $est_j = min(t)$. Hence all processes that start the last round have the same estimate $min(t)$ and then no other value can be decided. It contradicts the fact that (by assumption) some process decides a value different from $min(t)$. This contradiction concludes the proof of item (ii) for the base case.

Let us now consider the induction step. So, assuming that the items (i) and (ii) are satisfied from the first round until round $r-1$, let us show that they are satisfied for round $r \leq t+1$.

- Proof of (i): $|R(r)| \geq 1$. Let p_i be a process that terminates round r and such that $est_i \leq min(t)$ at the end of round r. Due to the definition of $min(t)$ and the fact that there is no creation of value, such a process necessarily exists. There are two cases.
  Case 1: $p_i \in Q(r)$. As p_i terminates round r, and no process in $R(r-1)$, $R(r-2)$, ..., $R(1)$ terminates round r (induction assumption), it follows that $p_i \notin R(0) \cup R(1) \ldots R(r-1)$. By the definition of $R(r)$, it follows that $p_i \in R(r)$. Hence, $|R(r)| \geq 1$, which proves item (i) for the case.
  Case 2: $p_i \notin Q(r)$. As $p_i \notin Q(r)$, it follows that p_i has received a message EST (r, est_j) from a process p_j such that $est_j \leq min(t)$ at the beginning of r, i.e., $p_j \in Q(r)$.

Claim C3 : $p_j \notin Q(r-1)$.
Assuming C3, as $p_j \notin Q(r-1)$ and due to the fact that est_j can only decrease during an execution, $p_j \notin Q(r')$ for all $r' < r$. Since $p_j \in Q(r)$ we have $p_j \in R(r)$, and consequently $|R(r)| \geq 1$, which proves item (i) for Case 2.

Proof of the claim C3. As p_i has received EST (r, est_j) with $est_j \leq min(t)$ from process p_j during round r, we conclude from the monotonicity property of the set $crashed_i$, that p_i has received a message EST $(r-1, -)$ from process p_j during round $r-1$. Let us assume by contradiction that $p_j \in Q(r-1)$. It then follows that p_i has received EST $(r-1, est_j)$ from p_j during round $r-1$ with $est_j \leq min(t)$, and consequently we have $est_i \leq min(t)$, at the end of round $r-1$, i.e., $p_i \in Q(r)$ which contradicts the assumption associated with Case 2. Hence, $p_j \notin Q(r-1)$. End of proof of claim C3.

- Proof of (ii) of claim C2: all the processes in $R(r)$ crash while executing round r or round $r+1$. The proof of this item is verbatim the same as the proof for the base case after changing the round numbers 1 and 2 by the round numbers r and $r+1$, respectively. So, it is not repeated here. End of proof of claim C2.

$\square_{Theorem\ 1}$

3.3 An early deciding/stopping algorithm

Underlying principle The principle that underlies the proposed early-deciding/stopping algorithm (described in Figure 3) is as follows. A process decides as soon as it knows that (1) its current estimate of the decision value is the smallest estimate value present in the system, and (2) at least one non-faulty process knows that value. This algorithm extends the previous one as follows. A line tagged (x) or (x') corresponds to line (x) in the non early-deciding algorithm. The lines tagged (Ny) are new lines that ensure early decision/stopping.

Local variables In addition to the previous local variables, the algorithm uses the following local ones.

- i_knows_i is a boolean, initialized to *false*. This boolean is set to *true* at the end of a round r, if p_i learned -during that round- that its current estimate est_i contains the smallest estimate value among the processes that start round r. This means that est_i contains the smallest value still present in the system. This boolean is stable (once true, it remains true forever).
- $they\_know_i$ is a set, initialized to $\emptyset$. This set contains the identities of the processes that, to p_i's knowledge, have the smallest estimate value present in the system.

```
operation propose (v_i):
(1')  est_i ← v_i; r_i ← 1; they_know_i ← ∅; i_know_i ← false;
(2)   while r_i ≤ t + 1 do
(3)       begin asynchronous round
(4')      broadcast EST (r_i, est_i, i_know_i);
(5')      wait until (∀j ∉ ((crashed_i ∪ they_know_i) \ {i})): (EST (r_i, −, −) received from p_j);
(N1)      let crashed_or_knowing_i be the set (crashed_i ∪ they_known_i) when the wait terminates;
(6')      let rec_from_i = {1, ..., n} \ crashed_or_knowing_i;
(7)       let est_rec_i = {est received during r_i from the processes in rec_from_i};
(8)       est_i ← min(est_rec_i);
(N2)      they_know_i ← they_know_i ∪ {x | EST (r_i, −, true) rec. from p_x with x ∈ rec_from_i};
(N3)      if ((|crashed_i ∪ they_know_i| ≥ t + 1) ∧ i_know_i) then return (est_i) end if;
(N4)      let some_knows_i = (∃ EST (r_i, −, true) received from p_x with x ∈ rec_from_i);
(N5)      i_know_i ← (some_knows_i) ∨ (|rec_from_i| ≥ n − r_i + 1);
(9)       r_i ← r_i + 1
(10)      end asynchronous round
(11)  end while;
(12)  return (est_i).
```

Figure 3: Early deciding/stopping consensus in $\min(f+2, t+1)$ rounds in $\mathcal{AS}_{n,t}[P]$ (code for p_i)

Behavior of a process As in the previous algorithm, the behavior of a process p_i during a round r is made up of two phases, a communication phase followed by a local computation phase.

- Communication phase. A process p_i first broadcasts EST (r_i, est_i, i_know_i) to inform the other processes on its current state. Then, it waits until it has received a message EST $(r_i, -, -)$ from each process p_j but the processes that, to its knowledge, have crashed or know the smallest estimate value. Moreover, we assume that it always receives its own message (this is not necessary simplifies the proof). Now, rec_from_i denotes the set of processes from which p_i considers and processes the messages it has received (those are the messages from itself and the processes not in $crashed_i \cup they\_known_i$).

  Let us observe that a process that knows the smallest estimate value can have decided and stopped in a previous round, and consequently, if p_i was waiting a message from it, it could be blocked forever.

- Computation phase. Then p_i computes its new estimate value, that is the smallest value carried by a message it has received from a process in rec_from_i. It also updates the set $they\_knows_i$ according to the boolean values carried by the EST $(r_i, -, -)$ messages it has received from the processes in rec_from_i. Then p_i strives to early decide. To that end, it does the following.

  - If it knows the smallest among the estimate values of the processes that start round r (i.e, i_knows_i is true), and knows also that that value is know by at least one non-faulty process (i.e., $|crashed_i \cup they\_know_i| \geq t+1$), then p_i decides and stops its participation to the algorithm.
  - Otherwise, p_i updates its local state before starting a new round. To that end, it has first computes the value of i_knows_i. If p_i has received a message EST $(r_i, v, true)$, it has learned from another process that v is the smallest estimate value, and consequently sets i_knows_i to $true$.

    As we will see in the proof, if $|rec\_from_i| \geq n - r + 1$, then p_i discovers that the current value of its estimate is the smallest estimate value in the system. In that case, it sets i_knows_i to the value $true$.

3.4 Proof of the algorithm

Theorem 2 *Let f be the number of processes that crash in a run ($0 \leq f \leq t$). The algorithm described in Figure 3 solves consensus in at most* $\min(f+2, t+1)$ *rounds in $\mathcal{AS}_{n,t}[P]$.*

Notation Let xxx_i be a local variable of process p_i. It is easy to see, that such a variable is updated at most once in a round r. The notation $xxx_i(r)$ is used to denote its value at the end of round r.

Proof The proof of the validity property is the same as in Theorem 1. Before proving the other properties, a few claims are proved.

Claim C1. $i\_know_i(r) \Rightarrow (est_i(r) = min(r-1))$ (i.e., at the end of round r, p_i knows the smallest estimate value that was present in the system at the end of $r-1$).
Proof of claim C1. The proof is by induction on the round number.

- If a process p_i updates its boolean i_know_i to true during round 1 (we have then $|rec\_from_i| = n$), it has received n messages containing the n initial estimates. Hence p_i knows the minimum value in the system amongst all the estimate values.

- Let p_i be a process that updates its boolean i_know_i to true during round r. There are two cases.

  - p_i has received a message EST $(r, -, true)$ from p_j. It means that p_j has updated its boolean i_know_j to true in a previous round, and then by induction, p_i knows the minimum value.
  - p_i has received $n-r+1$ messages in round r and it has never received (in round r or in previous rounds) a message EST $(-, -, true)$. For p_i's point of view, the execution is similar to an execution of the algorithm described in Figure 2 in which we would have $t = r-1$. This follows from the fact that (a) line N5 updates i_know_i only in round r, and (b) lines N1, N2, N3, and N4 do not modify local variables up to round r (they are consequently useless up to that round). Moreover, due to the current values of $i\_know_i = false$ and $they\_know_i = \emptyset$ before line N5 of round r, (1) the lines tagged (x') behave as the line tagged (x) in the algorithm of Figure 2, and (2) we have $|rec\_from_i(r)| \geq n - r + 1$. Consequently, at the end of r, i_knows_i equal to true, and it follows from Theorem 1 that we have then $est_i(r) = min(r-1)$ (smallest estimate value of processes that terminate round $r-1$). End of proof of claim C1.

Claim C2. No process blocks forever in a round.
Proof of claim C2. Let us assume by contradiction that processes block forever in a round. Let r be the first round during which a process p_i blocks forever. This happens in the **wait** statement where p_i waits for messages EST $(r, -, -)$ from the processes that currently are not in $crashed_i \cup they\_know_i$. If a process p_j crashes, it eventually appears in the set $crashed_i$ (let us remember that this set can only increase). Hence, a process that crashes cannot prevent p_i from progressing. If p_j is non-faulty, there are two cases.

- Case 1: p_j has not decided during a round $r' \leq r$. In that case, as by assumption r is the first round during which processes block forever, it follows that p_j sends a message EST $(r, -, -)$. Consequently, such a process p_j cannot block forever p_i in round r.

- Case 2: p_j has decided during a round $r' < r$. Before deciding during r', p_j sent EST $(r', -, true)$ to p_i, and (as p_j does not crash) p_i received and processed this message. Consequently, $j \in they\_knows_i(r')$. As the set $they\_knows_i$ is a non-decreasing set, and $r' < r$, it follows that $j \in they\_knows_i(r-1)$ (that is the value of $they\_knows_i$ when p_i waits during round r). Hence, in that case also, pj cannot block forever p_i. End of proof of claim C2.

Proof of the agreement property. Let r be the first round during which a process (say p_i) decides. Due to Claim C1, we have $est_i(r) = min(r-1)$. If p_j decides at round r, we also have $est_j(r) = min(r-1)$, from which we conclude that the processes that decide at round r, decide the same value.

Let p_k be a process that proceeds to round $r+1$. Let us observe that, when p_i decides, we have $|crashed_i \cup they\_know_i| \geq t+1$, from which we conclude that there is a non-faulty process p_x that sent EST $(r', est_x(r'-1), i\_knows_x(r'-1))$ (with $i\_knows_x(r'-1) = true$) during round $r' < r$. As p_x is non-faulty, it follows that every non-crashed processes received this message during round r'. Due to Claim C1, we have $i\_knows_x(r'-1) \Rightarrow \big(est_i(r'-1) = min(r'-2)\big)$. (Notice that $r' > 1$ because any i_know_x is initially equal to *false*, hence $r' \geq 2$.

As p_i received $(r', est_x(r'-1), true)$, it follows that $est_i(r) = est_x(r'-1)$. Consequently, any process p_k that proceeds to the round $r+1$ is such that $est_k(r) = est_x(r'-1) = est_i(r)$, i.e., the estimate values of all the processes that proceed to round $r+1$ are equal to $est_i(r)$. It follows that no value different from $est_i(r)$ can be decided in a round $r'' \geq r$, which completes the proof of that agreement property.
Proof of the termination property. Due to Claim C2, no process blocks forever in a round. If follows that if a process neither crashes, nor decides at a round $r < t+1$, it proceeds to round $t+1$ during which it decides (in the worst case at the last line of the algorithm).
Proof of the early decision property. Let us first observe that a process executes at most $t+1$ rounds. Hence, considering that $f < t$ processes crash, we have to show that no process decides after round $(f+2)$. There are two cases.

- Case 1: a correct process p_c sends EST $(r, -, true)$, during a round $r \leq f+1$. As p_c is non-faulty, every process that executes round r receives this message during round r. It follows that we have $i\_knows_i(r) = true$ at each process p_i that terminates round r. Consequently, every process p_j that executes round $r+1$ ($\leq f+2$) sends EST $(r+1, -, true)$, and hence we have $|crashed_j(r+1) \cup they\_know_j(r+1)| = n$. It follows that the early decision predicate is satisfied, and every process that execute round $r+1$ decides (and stops).

- Case 2: no correct process sends EST $(r, -, true)$ during a round $r \leq f+1$. In that case, no correct process appears in a set $they\_know_x(r)$ for $r \leq f+1$. Moreover, As f processes crash, and no correct process has decided, at least $(n-f)$ processes send messages during each round. Hence, at any round, we have $|rec\_from_i| \geq n-f$.

  Any process p_i that, during round $f+1$, evaluates the predicate $|rec\_from_i| \geq n-r+1$ finds that it is satisfied (because we have then $|rec\_from_i| \geq n-f = n-r+1$). Consequently, every process p_i that terminates round $f+1$, sets i_knows_i to true.

  Then, for any process p_i that does not crashes during $r = f+2$, and any process p_j, we have $j \in crashed_i$ or $j \in rec\_from_i$ when p_i terminates its **wait** statement. Moreover, if $j \in rec\_from_i$, p_i has received EST $(f+2, -, true)$ from p_j. It follows that $|crashed_i(f+2) \cup they\_know_i(f+2)| = n$. Consequently, during round $f+2$, p_i executes return(), i.e., it decides and stops.

$\square_{Theorem\ 2}$

3.5 Synchronous vs asynchronous system

In a synchronous model, it is possible to design an early-deciding algorithm similar to the one described in Figure 3. In such an algorithm, synchrony makes useless the set $they\_know_i$, and consequently the early decision predicate $(|crashed_i \cup they\_know_i| \geq t+1) \wedge i\_know_i$ (line N3) reduces to i_know_i.

In a synchronous model, instead of being $P1 = (|rec\_from_i| \geq n - r + 1)$, the predicate (used at line N5) that allows a process to discover that it knows the smallest estimate, can be $P2 = (prev\_nb_i = cur\_nb_i)$, where $prev\_nb_i$ and cur_nb_i are the number of messages that p_i has received during the previous round and the current round, respectively (with $prev\_nb_i$ is initialized to n) [17]. Interestingly, the predicate $P2$ involves two consecutive rounds $r-1$ and r, whatever the value of r. Differently, the predicate $P1$ involves explicitly the value of r. Moreover, it is easy to show that ($P1$ satisfied at round r) $\Rightarrow$ ($P2$ satisfied at some round $r' \leq r$), while we do not necessarily have ($P2$ satisfied at some round r') $\Rightarrow$ ($P1$ satisfied at the same round r'), which means that, in some failure patterns, $P2$ allows deciding earlier than $P1$ [16].

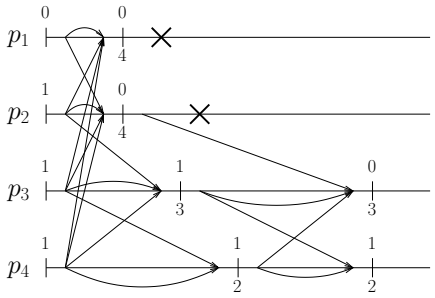

Figure 4: On early decision predicates

Unfortunately, in an asynchronous system enriched with a perfect failure detector, the predicate $P2$ does not allow p_i to conclude that it knows the smallest value present in the system. To show it, let us consider the counter-example depicted on Figure 4, where $n = 4$ and $t = 2$. The value proposed by a process p_i appears at the beginning of its time line (p_1 proposes 0, the other processes propose 1). At the end of a round, the integer above a process time line represents its new estimate value, while the integer below denotes the number of processes from which it has received messages. For example, during the second round, p_3 has received 3 messages, and its estimate is 0 at the end of this round. Finally, as indicated by crosses, p_1 and p_2 crash. It is easy to see that, while p_4 receives the same number of messages (namely, 2) during the first two rounds, it is not aware of the smallest value present in the system at the end of the second round (namely, 0).

4 A Theta-based perfect failure detector algorithm

4.1 The Theta model

The Theta asynchronous message-passing model has been proposed in [18]. It has been used to implement failure detectors in [2, 14]. We introduce it here incrementally and informally (for more details, the reader will consult [18]). The Theta model is actually a framework of models based on the following feature.

Considering a run of a synchronous system, let δ^+ (resp., δ^-) be the maximal (resp. minimal) transit time for a message between any two distinct processes. Let $\theta \geq \lceil \frac{\delta^+}{\delta^-} \rceil$. As we can see, θ actually characterizes an infinite set of runs, $R_1, R_2, \ldots$, each run R_x with its own pair of bounds (δ_x^+, δ_x^-) such that $\theta \geq \lceil \frac{\delta_x^+}{\delta_x^-} \rceil$.

Let us now consider an infinite run of an asynchronous system, where, while there are no bounds δ^+ and δ^- on message transfer delays, the run can be sliced in consecutive periods such that, during each period, θ is greater than or equal to the ratio of the maximal and the minimal transit times that occur during that period. As an example, this appears when both the maximal and the minimal transit times doubles from one period to the next one.

An instance of the Theta model is the asynchronous message-passing model in which all the runs are characterized by the constant ratio θ. It is important to notice that, while the value of θ is known, (if they exist) the values of δ^+ and δ^- are never known. In a very interesting way, the definition of this model does refer (directly) to physical time. This instance is denoted $\mathcal{AS}_{n,t}[\theta]$.

Another instance of the Theta model is when the bound θ holds only eventually. We denote this instance $\mathcal{AS}_{n,t}[\diamond\theta]$. Both instances are considered below.

4.2 Building a perfect failure detector in the θ model

As we are about to see, θ captures enough synchrony in order to implement a perfect failure detector, while hiding to the processes the uncertainty created by message transfer delays.

The principle of the algorithm is as follows. Each process p_i monitors each other process p_j and suspects it when, assuming p_j is alive, its behavior would falsify the assumption on messages speeds captured by the constant θ. The algorithm is described in Figure 5.

```
init r_i ← 1; suspected_i;
     for each j ≠ i do send PING () to p_j end for.

when PING () is received from p_j: send PONG () to p_j.

when PONG () is received from p_j:
     for each k ≠ j do
       if (k ∉ suspected_i) then
           count_i[j,k] ← count_i[j,k] + 1;
           if (count_i[j,k] > θ)
               then suspected_i ← suspected_i ∪ {k}
               else count_i[k,j] ← 0
           end if
       end if
     end for;
     send PING () to p_j.
```

Figure 5: Building P in $\mathcal{AS}_{n,t}[\theta]$ (code for p_i)

First create a message exchange pattern A process p_i executes a sequence of rounds (without using explicit round numbers) with respect to each other process. During each round with respect to p_j, process p_i sends it a message PING () and waits for the PONG () message that p_j echoes when it receives PING () from p_i. Finally, when it receives this echo message, p_i starts a new round with respect to p_j by sending it a new PING () message. Let us observe that messages (do not carry a round number and) have a constant size. An example of message pattern generated by the algorithm is depicted in Figure 6 (where the messages between p_i and p_j take δ^- times units, while the ones between p_i and p_k take δ^+ times units).

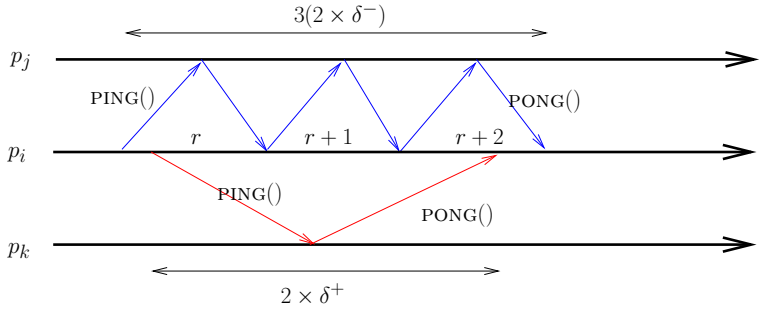

Figure 6: Example of message pattern in the θ model with $\theta = 3$

The conjunction of the assumption on message speeds captured by the constant θ and the PING/PONG messages generated as described previously actually makes the communication pattern satisfy the following α-*fairness* property [1].

- Communication is α-*fair* if, for any pair (p_i, p_j), process p_i receives at most α messages from process p_j without having received at least one message from each other non-crashed process.

It is easy to see that fair communication with $\alpha = 1$ is similar to the classical (synchronous) round-based lock-step system model, where in each round a process sends a message to each other process p_j and receives a message from each other non-crashed process p_j. More generally, in our asynchronous context, the PING/PONG messages exchanged between the processes create a θ-*fair* communication pattern.

Then exploit the fairness of the communication pattern In order to benefit from the θ-fairness property on the PONG/PONG messages exchanged by the processes, each process p_i manages a local array variable denoted $count_i[1..n, 1..n]$, the meaning of which is the following:

- $count_i[j,k] = x$ means that x messages from p_j have been received by p_i since the last message p_i has received from p_k.

The set $suspected_i$ built by the algorithm at every process p_i is initialized to $\emptyset$. When p_i receives a message PONG() from p_j, it does the following with respect to each process p_k that it does not suspect. If first increases $count_i[j,k]$. Then, it

checks the predicate $count_i[j,k] > \theta$. If this predicate is true, p_i has received more than θ messages from p_j without having received a message from p_k. As this would contradict the fair communication assumption if p_k was alive, p_i concludes that p_k has crashed. Consequently it adds the identity k to its set $suspected_i$. If the predicate is false, p_i resets $count_i[k,j]$ to 0, as from now on, it has received no PONG() message from p_k since the last PONG() message from p_j.

Theorem 3 *Let us assume that there are at least two non-faulty processes. The algorithm described in Figure 5 builds a perfect failure detector in the θ model. Moreover, the algorithm uses only bounded variables.*

Proof
Proof of the completeness property. The complete-ness property of the class P follows from the fact that a process p_k that crashes is discovered faulty by p_i because there is at least another non-faulty process p_j. More precisely, after p_k has crashed, it no longer sends messages and consequently there is a finite time from which $count_i[j,k]$ is never reset to 0. On another side, as p_j is non-faulty, it always answers by return the PING() messages it receives from p_i. Consequently, after some finite time, the local predicate $count_i[j,k] > \theta$ becomes true and p_i adds k to $suspected_i$. Finally, let us observe that, once added to $suspected_i$, no process identity is withdrawn from this set, which completes the proof of the completeness property.
Proof of the strong accuracy property. As there are always messages exchanged between alive processes, it follows from the θ assumption on the maximal ratio on the speed of messages that, when p_i, p_j and p_k are alive, p_i receives at most θ messages from p_j without receiving a message from p_k, which means that communication is θ-fair. It follows that, until p_k crashes if ever it does, the predicate $count_i[j,k] > \alpha$ is always false when p_i receives a message from any process p_j.

Let us finally observe that the value of a counter $count_i[j,k]$ varies between 0 and $\theta+1$, which establishes the boundedness property. $\square_{Theorem\ 3}$

4.3 From $\diamond\theta$ to $\diamond P$

A failure detector of the class $\diamond P$ can suspect processes before they crash during an arbitrarily long period, after which it behaves as a perfect failure detector.

The $\diamond\theta$ property states that there is a finite time (but unknown) after which the ratio of the upper and lower bounds on messages transfer delays is bounded by θ.

The algorithm described in Figure 7 builds a failure detector of the class $\diamond P$ in an asynchronous system that satisfies the $\diamond\theta$ property. This algorithm is a simple adaptation of the one described in Figure 5. The set $suspected_i$ can now momentarily contain non-crashed processes. This algorithm can be modified to take into account the case where, while the bound θ does exist, it is not known (however the local variables can no longer be bounded).

```
init r_i ← 1; suspected_i;
     for each j ≠ i do send PING () to p_j end for.

when PING () is received from p_j: send PONG () to p_j.

when PONG () is received from p_j:
     if (j ∈ suspected_i)
       then suspected_i ← suspected_i \ {j} end if;
     for each k ≠ j do
       if (k ∉ suspected_i) then
         count_i[j,k] ← count_i[j,k] + 1;
         if (count_i[j,k] > θ)
           then suspected_i ← suspected_i ∪ {k}
         end if;
       end if;
       count_i[k,j] ← 0
     end for;
     send PING () to p_j.
```

Figure 7: Building $\diamond P$ in $\mathcal{AS}_{n,t}[\diamond\theta]$ (code for p_i)

5 Conclusion

This paper has presented two contributions related to fault-tolerance in asynchronous systems. The first is an algorithm that builds an early-deciding *and* stopping consensus algorithm in an asynchronous system enriched with a perfect failure detector. The second is the construction of such a perfect detector in the Theta distributed computing model. Hence, the stacking of these algorithms provides an algorithm that solves consensus in the Theta model in $\min(f+2, t+1)$ communication rounds, i.e., in two rounds when there are no failures, which is clearly optimal.

From a technical point of view, it is worth noticing that the proof of the early-deciding and stopping consensus algorithm is far from being trivial. The feasibility of a proof as simple as the proof of early-deciding consensus algorithms in synchronous systems remains a challenging problem.

References

[1] Beauquier J. and Kekkonen-Moneta S., Fault-tolerance and Self-stabilization: Impossibility Results and Solutions Using Self-stabilizing Failure Detectors. *Int'l Journal of Systems Science*, 28(11):1177-1187, 1997.

[2] Biely M. and Widder J., Optimal Message-driven Implementations of Omega with Mute Processes. *ACM Transactions on Autonomous and Adaptive Systems*, 4(1), 22 pages, 2009.

[3] Chandra T., Hadzilacos V. and Toueg S., The Weakest Failure Detector for Solving Consensus. *JACM*, 43(4):685-722, 1996.

[4] Chandra T. and Toueg S., Unreliable Failure Detectors for Reliable Distributed Systems. *JACM*, 43(2):225-267, 1996.

[5] Charron-Bost B., Guerraoui R. and Schiper A., Synchronous System and Perfect Failure Detector: Solvability and Efficiency Issue. *Proceedings of DSN'00*, IEEE Computer Press, pp. 523-532, 2000.

[6] Dolev D., Reischuk R. and Strong R., Early Stopping in Byzantine Agreement. *JACM*, 37(4):720-741, 1990.

[7] Fischer M.J., Lynch N.A. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374-382, 1985.

[8] Fischer M.J. and Lynch N.A., A Lower Bound for the Time to Ensure Interactive Consistency. *Information Processing Letters*, 14:183-186, 1982.

[9] Guerraoui R. and Raynal M., The Information Structure of Indulgent Consensus. *IEEE Transactions on Computers*, 53(4):453-466, 2004.

[10] Guerraoui R. and Raynal M., The Alpha of Indulgent Consensus. *The Computer Journal*, 50(1):53-67, 2007.

[11] Keidar I. and Rajsbaum S., A Simple Proof of the Uniform Consensus Synchronous Lower Bound. *Information Processing Letters*, 85:47-52, 2003.

[12] Keidar I. and Shraer A., How to Choose a Timing Model. *IEEE Transsactions on Parallel Distributed Systems*, 19(10):1367-1380, 2008.

[13] Lamport L., The Part-time Parliament. *ACM Transactions on Computer Systems*, 16(2):133-169, 1998.

[14] Le Lann G. and Schmid U., How to Implement a Timer-free perfect Failure Detector in Partially Synchronous Systems. *TR 28/2005*, Institut Für Technische Informatik, TU Wien, 2005.

[15] Mostefaoui A. and Raynal M., Leader-Based Consensus. *Parallel Processing Letters*, 11(1):95-107, 2001.

[16] Raipin Parvedy Ph., Raynal M. and Travers C., Early-Stopping k-Set Agreement in Synchronous Systems Prone to Any Number of Process Crashes. *Proc. 8th Int'l Conf. on Parallel Computing Technologies (PaCT'05)*, Springer-Verlag LNCS #3606, pp. 49-58, 2005.

[17] Raynal M., Consensus in Synchronous Systems: a Concise Guided Tour. *9th IEEE Pacific Rim Int'l Symposium on Dependable Computing (PRDC'2002)*, IEEE Computer Press, pp. 221-228, 2002.

[18] Widder J. and Schmid U., The Theta-Model: Achieving Synchrony Without Clocks. *Distrbuted Computing*. 22(1):29-47, 2009.